

Санкт-Петербургский Политехнический Университет Петра Великого  
Институт компьютерных наук и технологий  
Кафедра компьютерных систем и программных технологий

# Программирование

Отчет по курсовой работе

Игра: Сапёр

**Работу выполнила:**

Кузовкина Е.О.

Группа: 23501/4

**Преподаватель:**

Вылегжанина К.Д.

Санкт-Петербург  
2017

# Содержание

<b>1</b>	<b>Игровое приложение: Сапёр</b>	<b>2</b>
1.1	Концепция игрового приложения Сапёр . . . . .	2
1.2	Задание . . . . .	2
1.3	Минимально работоспособный продукт . . . . .	2
1.4	Вывод . . . . .	2
<b>2</b>	<b>Проектирование игрового приложения Сапёр</b>	<b>2</b>
2.1	Архитектура приложения . . . . .	2
2.2	Диаграмма компонентов . . . . .	4
2.3	Формат задания игры . . . . .	4
2.4	Вывод . . . . .	4
<b>3</b>	<b>Реализация игрового приложения Сапёр</b>	<b>4</b>
3.1	Используемые версии . . . . .	4
3.2	Библиотека Swing . . . . .	4
3.3	Процесс разработки игрового приложения . . . . .	4
3.4	Перспективы развития приложения . . . . .	10
3.5	Вывод . . . . .	10
<b>4</b>	<b>Процесс обеспечения качества и тестирование игрового приложения Сапёр</b>	<b>10</b>
4.1	Тестирование . . . . .	10
4.2	Вывод . . . . .	10
<b>5</b>	<b>Выводы</b>	<b>10</b>
<b>6</b>	<b>Приложение</b>	<b>11</b>
6.1	Листинги . . . . .	11

# 1 Игровое приложение: Сапёр

«Сапёр» — компьютерная игра-головоломка.

Принцип игры:

Плоское или объёмное игровое поле разделено на смежные ячейки (квадраты, шестиугольники, кубы и т. п.), некоторые из которых «заминированы»; количество «заминированных» ячеек известно. Целью игры является открытие всех ячеек, не содержащих мины.

Игрок открывает ячейки, стараясь не открыть ячейку с миной. Открыв ячейку с миной, он проигрывает. Мины расставляются после первого хода, поэтому проиграть на первом же ходу невозможно. Если под открытой ячейкой мины нет, то в ней появляется число, показывающее, сколько ячеек, соседствующих с только что открытой, «заминировано» (в каждом варианте игры соседство определяется по-своему); используя эти числа, игрок пытается рассчитать расположение мин, однако иногда даже в середине и в конце игры некоторые ячейки всё же приходится открывать наугад. Если под соседними ячейками тоже нет мин, то открывается некоторая «не заминированная» область до ячеек, в которых есть цифры. «Заминированные» ячейки игрок может пометить, чтобы случайно не открыть их. Открыв все «не заминированные» ячейки, игрок выигрывает.

## 1.1 Концепция игрового приложения Сапёр

Программа представляет собой головоломку, которая позволяет нажимать на ячейки поля, открывая их или обозначая флажком, если игрок считает, что там находится бомба, или вопросительным знаком, если игрок предполагает, что эти ячейки не стоит открывать и там может быть бомба.

Приложение отрисовывает игровое поле, на котором располагаются бомбы. Пользователь не видит их. Пользователь может открывать любые ячейки и обновлять игру в любой момент, нажав на смайлик. Игра считается законченной, если пользователь разминировал все бомбы, то есть правильно расставил флажки.

## 1.2 Задание

Разработать приложение под операционные системы Windows 7+ и Android, позволяющее разгадывать головоломку Сапёр.

## 1.3 Минимально работоспособный продукт

Приложение, которое предоставляет возможность открывать ячейки и ставить флажки.

## 1.4 Вывод

Пояснён выбор темы курсового проекта. Описана концепция игрового приложения Сапёр. Определено задание.

# 2 Проектирование игрового приложения Сапёр

## 2.1 Архитектура приложения

Был использован шаблон проектирования Model-View-Presenter

Его использование обусловлено тем, что:

- требовалось обеспечить расширяемость приложения, так как существовала некоторая неопределённость по поводу того, какую функциональность должно предоставлять приложение, так как планировалось учесть новые пожелания пользователей
- требовалось обеспечить скорость разработки
- этот шаблон интересен с учебной точки зрения

Архитектура Модели выглядит следующим образом:

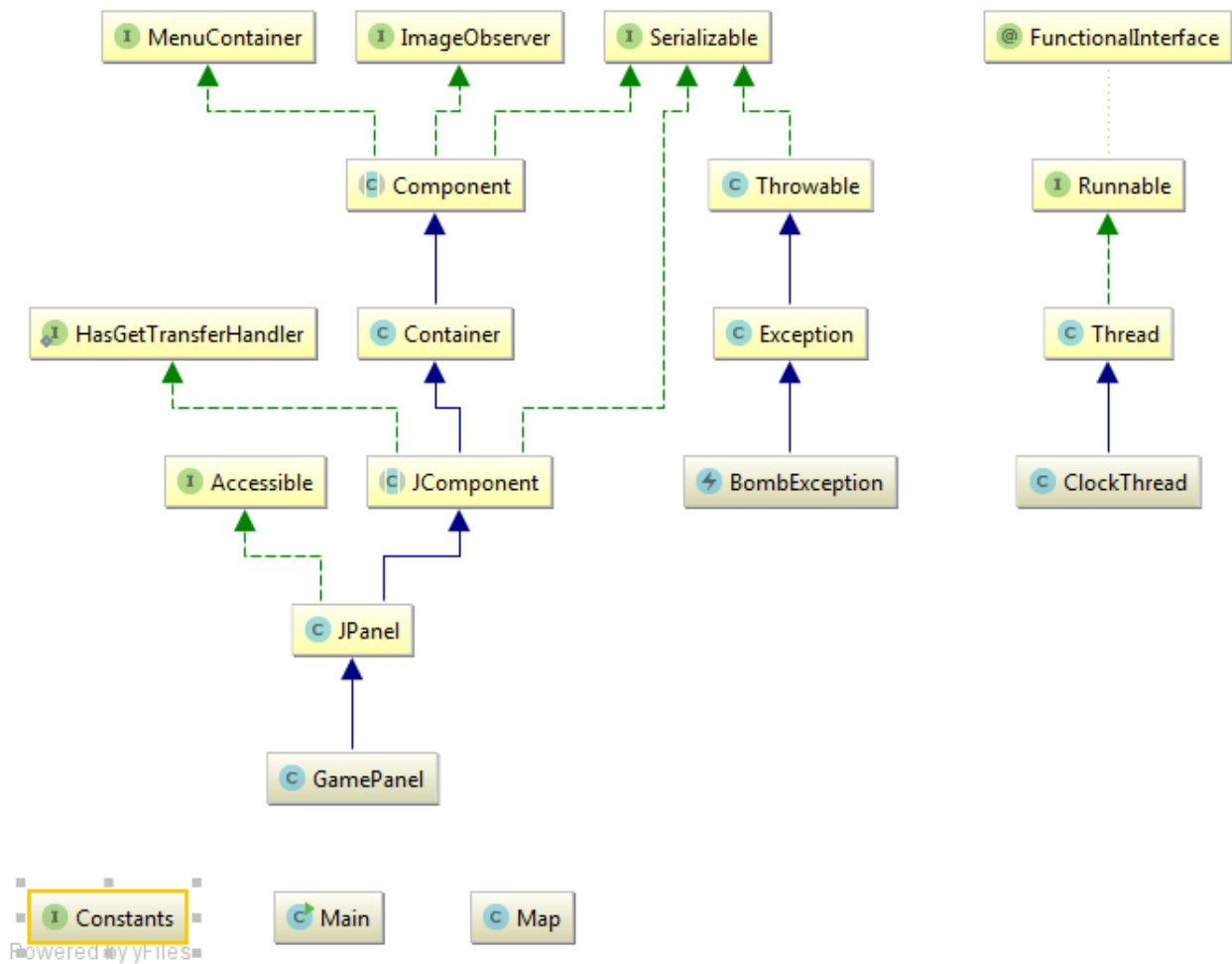


Рис. 1: Диаграмма классов Модели

Модель предоставляет следующую функциональность:

- Показать длительность игры
- Открыть ячейку
- Пометить ячейку вопросительным знаком
- Показать окончание игры, открытие всех ячеек
- Пометить ячейку флажком
- Установить игровое поле
- Получить ячейки поля
- Получить информацию о соседних ячейках
- Начать игру заново

## 2.2 Диаграмма компонентов

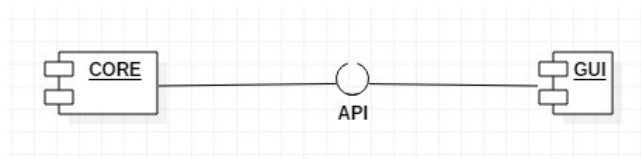


Рис. 2: Диаграмма компонентов

- Core – содержит логическую часть игры, предоставляет данные для пользовательского интерфейса.
- GUI – отвечает за взаимодействие с пользователем путём отрисовки изображения на экране и фиксирования команд и событий, которые впоследствии перенаправляет API.
- API – управляет CORE и GUI. Указывает GUI, что нужно отрисовывать в данный момент, принимает его оповещения о командах и сигналах, реагирует на них и, если это необходимо, связывается с CORE для получения данных.

## 2.3 Формат задания игры

Так как в игре 1 уровень, то пользователь может лишь обновлять поле для старта новой игры и он не может задавать размеры поля и количество мин. Для игры выбрано стандартное поле.

## 2.4 Вывод

Было решено использовать шаблон проектирования Observer и анти-паттерн Magic Numbers. Была описана функциональность предоставляемая Моделью. Был объяснён формат приложения.

# 3 Реализация игрового приложения Сапёр

## 3.1 Используемые версии

- IntelliJ IDEA 2016.3.1  
Build IU-163.9166.29  
For educational use only.  
JRE: 1.8.0 102-b14 amd64  
JVM: Java HotSpot(TM) 64-Bit Server VM by Oracle Corporation
- Java language level: 6
- Операционная система: Windows 7 x64
- Система автоматической сборки: Gradle 2.14

## 3.2 Библиотека Swing

В данном проекте было принято решение использовать библиотеку Swing.

Swing — библиотека для создания графического интерфейса для программ на языке Java. Он содержит ряд графических компонентов (Swing widgets), таких как кнопки, поля ввода, таблицы и т. д.

## 3.3 Процесс разработки игрового приложения

Было проведено первичное знакомство со Swing и создано приложение, в котором почти не участвует Модель. После получения опыта работы со Swing и принятия решения, что этот фреймворк подходит для решаемой задачи было решено заняться непосредственно развитием функциональности Модели. В итоге выбранный путь позволил корректировать Модель во время разработки таким образом, чтобы с ней было удобно работать.

На следующих изображениях поэтапно приведён процесс разработки приложения:

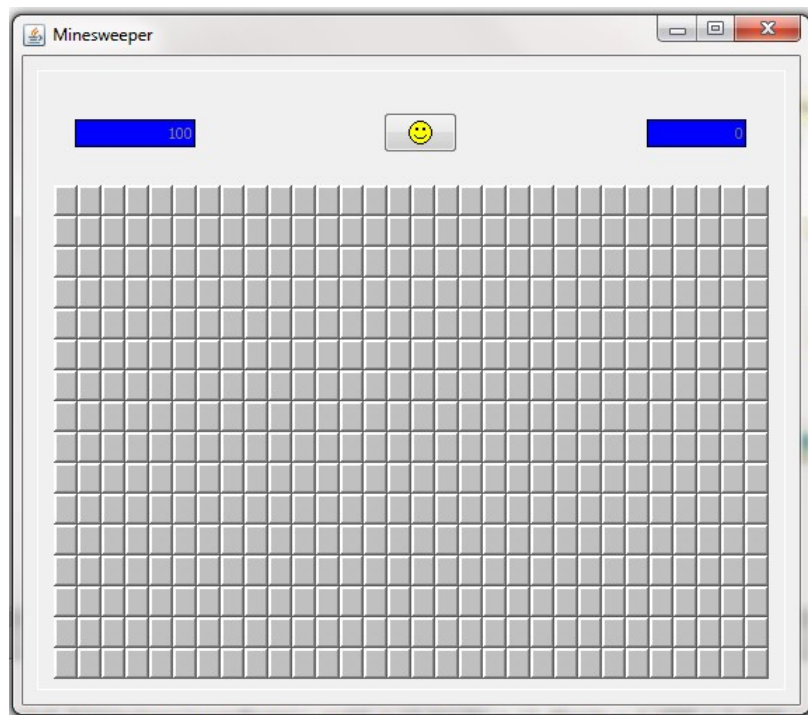


Рис. 3: Снимок экрана иллюстрирующий игровое окно, до начала игры

На рисунке 3 изображён экран, на котором отрисовано закрытое поле, два синих окна, в левом окошке - количество мин, которое необходимо обезвредить, в правом - время от начала игры в секундах. Время не начинает идти пока пользователь не нажмет на ячейку поля.

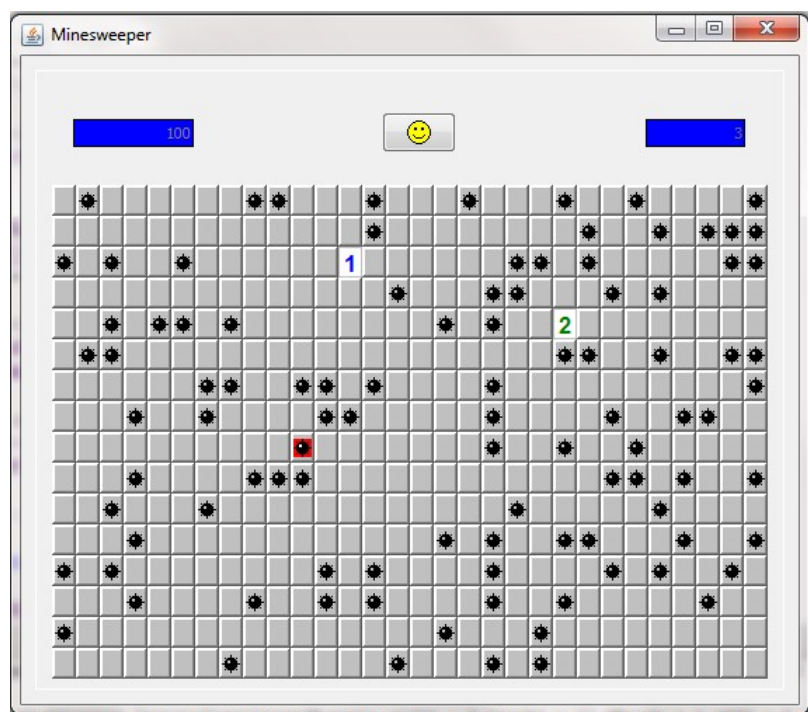
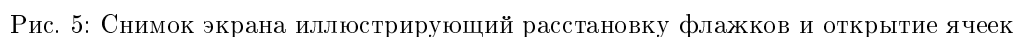


Рис. 4: Снимок экрана иллюстрирующий вид пользовательского интерфейса после попадания на мину

На рисунке 4 пользователь сделал три шага, на третьем шаге он открыл ячейку с миной и взорвался на ней, при этом мина, которая взорвалась обозначается красным, в отличие от остальных. Так как пользователь попал на мину, то открывается всё поле и игра заканчивается. Для продолжения игры необходимо

Для открытия ячеек поля пользователь использует левую клавишу мыши и один щелчок.



Для обозначения ячейки флажком, пользователь использует правую клавишу мыши и 1 щелчок. При этом ячейка остается закрытой, то есть неизвестно стоит ли под флажком мина или нет. Для отмены необходимо 2 раза нажать на ячейку, которую необходимо очистить.

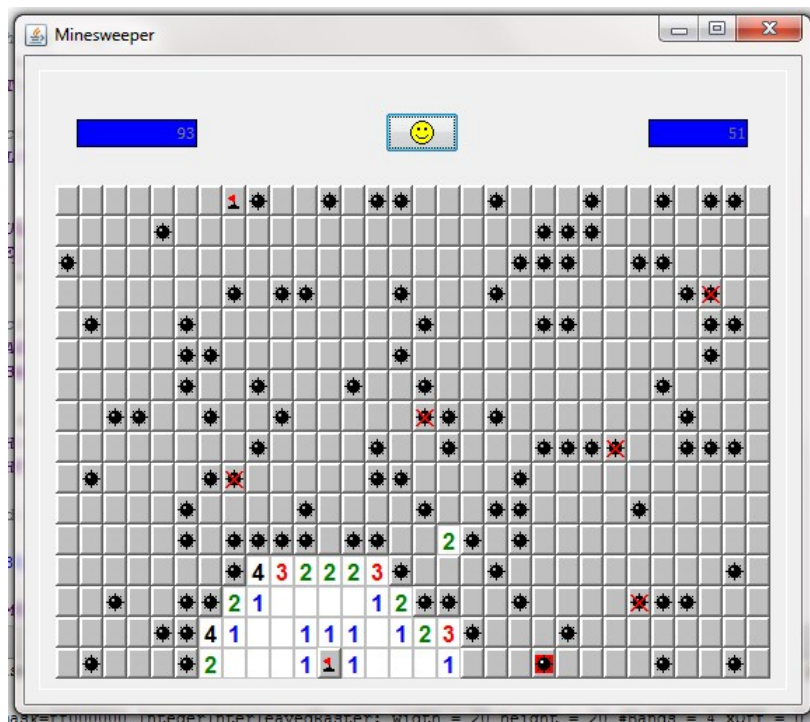


Рис. 6: Снимок экрана иллюстрирующий обезвреживание нескольких мин

На рисунке 6 пользователь попал на мину при открытии следующих ячеек. Те мины, которые были помечены флажком, показаны обезвреженными (зачеркнуты).

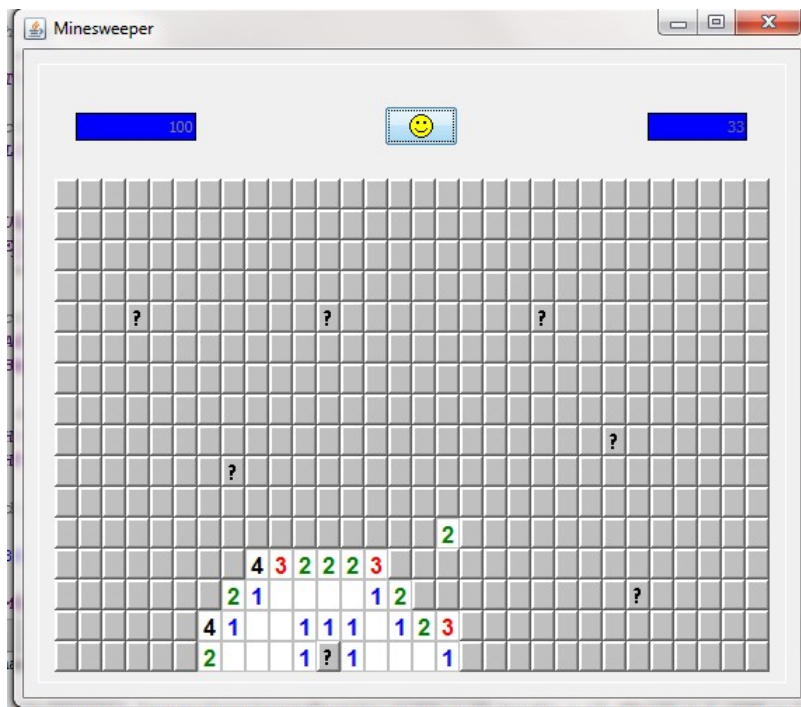


Рис. 7: Снимок экрана иллюстрирующий расстановку вопросительных знаков

На рисунке 7 пользователь расставил вопросительные знаки и открыл несколько ячеек. Вопросительный знак не обезвреживает мины, он лишь помогает пользователю, который сомневается, есть ли в данном месте мина.



Для обозначения ячейки вопросительным знаком, пользователь использует правую клавишу мыши и 2 щелчка. При этом ячейка остается закрытой. Для отмены необходимо еще 1 раз нажать на ячейку, с которой необходимо снять вопросительный знак.

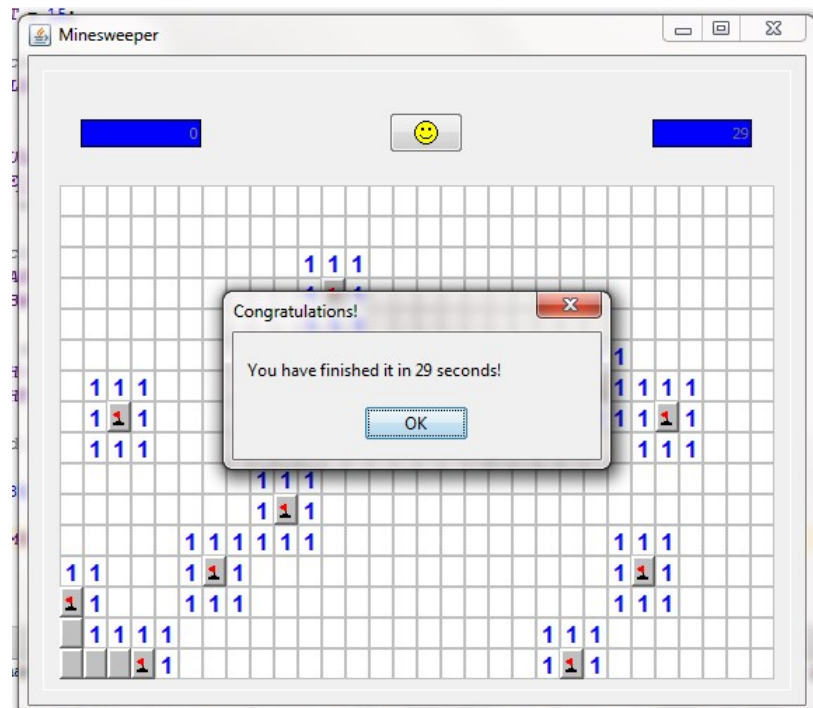


Рис. 8: Снимок экрана иллюстрирующий окончание игры

На рисунке 8 пользователь правильно разгадал головоломку и появился экран окончания игры с подсчитанным игровым временем (временем, затраченным на разгадку).

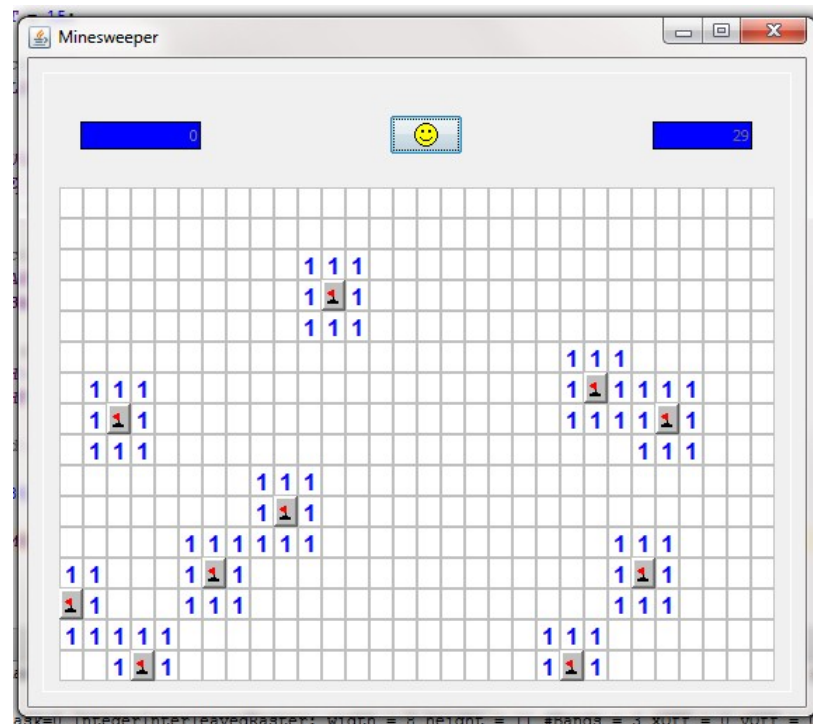


Рис. 9: Снимок экрана иллюстрирующий полностью открытое поле

На рисунке 9 пользователь закрыл окно, которое было на рис. 8 с поздравлениями с успешным прохождением головоломки и подсчитанным временем. После этого всё игровое поле открылось.

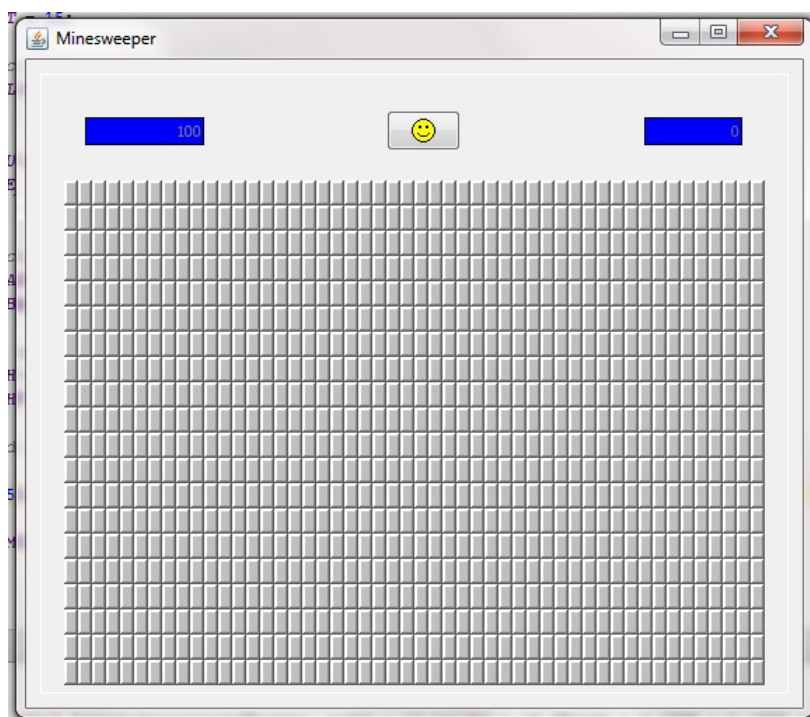


Рис. 10: Снимок экрана иллюстрирующий увеличенное игровое поле

На рисунке 10 показано увеличенное игровое поле.

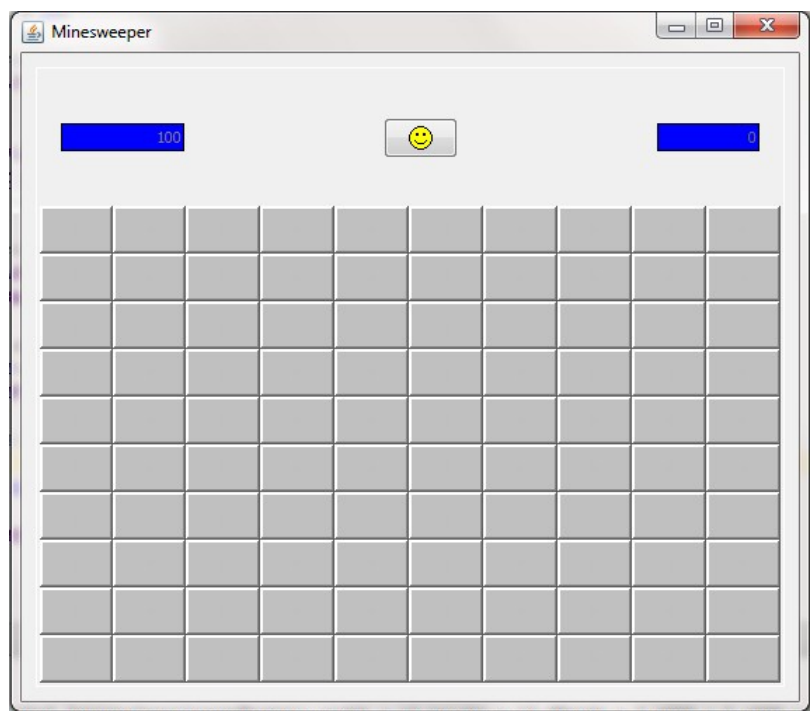


Рис. 11: Снимок экрана иллюстрирующий уменьшенное игровое поле

На рисунке 11 показано уменьшенное игровое поле.

В ходе разработки игры для удобства тестирования можно было изменять размеры поля и количество мин. Для окончательной игры было выбрано стандартное значение размеров поля и количество мин для удобства всех пользователей (и новичков, и любителей).

### 3.4 Перспективы развития приложения

Планируется реализовать следующую функциональность:

- добавить изменение размеров поля (пользователем)
- добавить изменение количества мин (пользователем)
- добавить окно поражения, появляющееся при взрывании на mine
- и т.п.

### 3.5 Вывод

Были описаны используемые средства разработки. Кратко описан фреймворк Swing. Был поэтапно описан процесс разработки приложения и пути дальнейшего развития приложения.

## 4 Процесс обеспечения качества и тестирование игрового приложения Сапёр

Для проверки корректности работы проводилось ручное тестирование.

### 4.1 Тестирование

Тестирование проводилось по следующему сценарию:

- 1 Запустить приложение, проверить отрисовку поля и окошек с количеством мин и игровым временем
- 2 Открыть любую ячейку поля, проверить, что в ней находится, есть ли цифры показывающие, где скорее всего находится мина
- 3 Если попала мина, то нажать на смайлик и начать игру заново (вернуться в п.2), проверить работает ли обновление игрового поля и старт новой игры
- 4 Если открылись ячейки, то рассчитать, где находится мина и поставить на ней флажок, пометить вопросительным знаком все сомнительные ячейки (те, в которых может быть мина, а может и ничего не быть), проверить ставятся ли флажки, вопросительные знаки
- 5 Открыть следующие ячейки, в которых скорее всего нет мины
- 6 При попадании на мину начать игру заново (вернуться в п.2), проверить открывается ли все игровое поле, чтобы показать расположение всех мин
- 7 Продолжить открытие ячеек (как описано в п.4-п.6)
- 8 В случае победы проверить правильность информации в окне поздравлений, закрыть его и попробовать начать игру заново, нажав на смайлик
- 9 Нажать на кнопку выхода и закрыть приложение

### 4.2 Вывод

Был описан процесс тестирования приложения.

## 5 Выводы

Было разработано игровое приложение Сапёр. Была изучена библиотека Swing и паттерн проектирования Model-View-Presenter. Созданное в ходе работы приложение было протестировано, также были определены возможные перспективы развития функциональности приложения. В дальнейшем планируется улучшение приложения, а также исправление возможных недочетов.

## 6 Приложение

Исходный код можно найти в репозитории<sup>1</sup> на ресурсе GitHub

### 6.1 Листинги

```
1 package minesweeper.logic;
2
3 import java.util.Random;
4
5 import minesweeper.Constants;
6
7 public class Map {
8
9     // Slots matrix
10    private Slot [][] matrix;
11
12
13    // Builds a populated Map
14    public Map() {
15
16        // Create the (empty) matrix
17        matrix = new Slot [ Constants.ROWS ][ Constants.COLUMNS ];
18        for ( int theRow = 0; theRow < Constants.ROWS; theRow++ ) {
19
20            for ( int theColumn = 0; theColumn < Constants.COLUMNS; theColumn++ )
21            {
22                // A new Slot for each position
23                matrix [ theRow ][ theColumn ] = new Slot ();
24            }
25        }
26        // Populate the matrix
27        populateMap ();
28    }
29
30    // Getters
31    public boolean isThereAMine ( int theRow, int theColumn ) { return matrix [ theRow ][ theColumn ].
    ↪ hasMine (); }
32    public boolean isHidden ( int theRow, int theColumn ) {
33        return matrix [ theRow ][ theColumn ]. isHidden ();
34    }
35    public boolean isSuspicious ( int theRow, int theColumn ) {
36        return matrix [ theRow ][ theColumn ]. isSuspicious ();
37    }
38    public boolean hasQuestionMark ( int theRow, int theColumn ) {
39        return matrix [ theRow ][ theColumn ]. hasQuestionMark ();
40    }
41
42    // Setters
43    public void setHasMine ( int theRow, int theColumn ) {
44        matrix [ theRow ][ theColumn ]. setHasMine ( true );
45    }
46
47    public void setHasBeenVisitedThisTurn ( int theRow, int theColumn, boolean value ) {
48        matrix [ theRow ][ theColumn ]. setHasBeenVisitedThisTurn ( value );
49    }
50
51    private void clear ( int theRow, int theColumn ) {
52        matrix [ theRow ][ theColumn ]. setIsHidden ( false );
53        matrix [ theRow ][ theColumn ]. setIsSuspicious ( false );
54        matrix [ theRow ][ theColumn ]. setHasQuestionMark ( false );
55    }
56
57    public void setSuspicious ( int theRow, int theColumn ) {
58        matrix [ theRow ][ theColumn ]. setIsSuspicious ( true );
59    }
60    public void setFreeOfSuspicion ( int theRow, int theColumn ) {
61        matrix [ theRow ][ theColumn ]. setIsSuspicious ( false );
62    }
63    public void addQuestionMark ( int theRow, int theColumn ) {
64        matrix [ theRow ][ theColumn ]. setHasQuestionMark ( true );
65    }
66    public void clearQuestionMark ( int theRow, int theColumn ) {
```

---

<sup>1</sup><https://github.com/kuzo-liza/Minesweeper>

```

67     matrix[theRow][theColumn].setHasQuestionMark(false);
68 }
69
70 // Clear the 'hasBeenVisited' flag in all the game grid
71 private void renewGameGrid() {
72
73     for (int theRow = 0; theRow < Constants.ROWS; theRow++) {
74         for (int theColumn = 0; theColumn < Constants.COLUMNS; theColumn++) {
75
76             // New turn, nothing has been visited
77             Slot slot = matrix[theRow][theColumn];
78             slot.setHasBeenVisitedThisTurn(false);
79             slot.setHasMine(false);
80             slot.setIsHidden(true);
81             slot.setIsSuspicious(false);
82             slot.setHasQuestionMark(false);
83         }
84     }
85 }
86
87 // Has the current game ended?
88 public boolean hasWon() {
89
90     // Is every bombed slot marked as suspicious? Is there any hidden slot left?
91     boolean ret = true;
92     int theRow = 0;
93
94     while ((theRow < Constants.ROWS) && ret) {
95         int theColumn = 0;
96
97         while (theColumn < Constants.COLUMNS && ret) {
98
99             Slot slot = matrix[theRow][theColumn];
100             // If there is any unmarked hidden slot, or a wrongly marked bombed slot, we are not
101             ↪ done yet
102             ↪ if ((slot.isHidden() && !slot.isSuspicious()) || (slot.isHidden() && slot.isSuspicious
103             ↪ () && !slot.hasMine())) {
104                 ret = false;
105             }
106             theColumn++;
107         }
108         theRow++;
109     }
110     return ret;
111 }
112
113 // Check if a player's click has stepped into a bomb
114 public void click(int theRow, int theColumn) throws BombException {
115     if (isSuspicious(theRow, theColumn)) {
116         // Nothing done. The 'suspicious' mark protects the slot
117         return;
118     }
119
120     // Bomb found? Then throw exception to be managed
121     if (matrix[theRow][theColumn].hasMine()) {
122         throw new BombException(theRow, theColumn);
123     }
124     else {
125         // Clear recursively the game board around the slot
126         clickI(theRow, theColumn);
127     }
128 }
129
130 // Recursive procedure to clear near slots
131 private void clickI(int theRow, int theColumn) {
132     Slot slot = matrix[theRow][theColumn];
133
134     // If it has a bomb, do nothing
135     // If it has been 'visited' this turn, we do nothing
136     // If it was detected, we do nothing
137     if (!slot.hasMine() && !slot.hasBeenVisitedThisTurn() && slot.isHidden()) {
138
139         // We visit it
140         slot.setHasBeenVisitedThisTurn(true);
141
142         // Call again for the adjacent slots (clockwise)

```

```

141     if (getMinesAround(theRow, theColumn) == 0) {
142
143         // Upper row
144         if (theRow > 0) {
145             if (theColumn > 0) {
146                 // Up left
147                 clickI(theRow - 1, theColumn - 1);
148             }
149
150             // Up
151             clickI(theRow - 1, theColumn);
152             if (theColumn < Constants.COLUMNS - 1) {
153                 // Up right
154                 clickI(theRow - 1, theColumn + 1);
155             }
156         }
157
158         // To the right
159         if (theColumn < Constants.COLUMNS - 1) { clickI(theRow, theColumn + 1); }
160
161         // Lower row
162         if (theRow < Constants.ROWS - 1) {
163             if (theColumn < Constants.COLUMNS - 1) {
164                 // Down-right
165                 clickI(theRow + 1, theColumn + 1);
166             }
167
168             // Down
169             clickI(theRow + 1, theColumn);
170             if (theColumn > 0) {
171                 // DownLeft
172                 clickI(theRow + 1, theColumn - 1);
173             }
174         }
175
176         // To the left
177         if (theColumn > 0) { clickI(theRow, theColumn - 1); }
178     }
179
180     // Reveal the slot
181     clear(theRow, theColumn);
182 }
183 }
184
185 // It clears the game board and populates it with a number of bombs in random locations
186 public void populateMap() {
187     renewGameGrid();
188
189     // Randomly place mines
190     int minesPlaced = 0;
191     Random g = new Random();
192
193     while (minesPlaced < Constants.NUMBER_OF_MINES) {
194
195         // Generate coordinates for the mine
196         int row = g.nextInt(Constants.ROWS);
197         int column = g.nextInt(Constants.COLUMNS);
198
199         // Place it if it was clear
200         // Maybe the coordinates are not valid, we need a new turn of the loop
201         if (!isThereAMine(row, column)) {
202             minesPlaced++;
203             setHasMine(row, column);
204         }
205     }
206 }
207
208 // Counts the number of remaining mines for the player. Beware that it must make its count
209 // ↪ with
210 // the number of flags that the player has placed.
211 public int remainingMines() {
212     return Constants.NUMBER_OF_MINES - countGuessedMines();
213 }
214
215 // It counts the number of slots marked as 'suspicious' by the player
216 private int countGuessedMines() {

```

```

216     int counter = 0;
217
218     for (int theRow = 0; theRow < Constants.ROWS; theRow++) {
219         for (int theColumn = 0; theColumn < Constants.COLUMNS; theColumn++) {
220
221             // If it is hidden, add 1 to the counter
222             if (matrix[theRow][theColumn].isSuspicious()) { counter++; }
223         }
224     }
225     return counter;
226 }
227
228 // Calculates the number of mines around a given slot
229 public int getMinesAround(int theRow, int theColumn) {
230     int ret = 0;
231
232     // Upper row
233     if (theRow > 0) {
234         if (theColumn > 0) {
235             // Up left
236             if (isThereAMine(theRow - 1, theColumn - 1)) { ret++; }
237         }
238
239         // Up
240         if (isThereAMine(theRow - 1, theColumn)) { ret++; }
241         if (theColumn < Constants.COLUMNS - 1) {
242             // Up right
243             if (isThereAMine(theRow - 1, theColumn + 1)) { ret++; }
244         }
245     }
246
247     // To the right
248     if (theColumn < Constants.COLUMNS - 1) {
249         if (isThereAMine(theRow, theColumn + 1)) { ret++; }
250     }
251
252     // Lower row
253     if (theRow < Constants.ROWS - 1) {
254         if (theColumn < Constants.COLUMNS - 1) {
255             // Down-right
256             if (isThereAMine(theRow + 1, theColumn + 1)) { ret++; }
257         }
258
259         // Down
260         if (isThereAMine(theRow + 1, theColumn)) { ret++; }
261         if (theColumn > 0) {
262             // Down-left
263             if (isThereAMine(theRow + 1, theColumn - 1)) { ret++; }
264         }
265     }
266
267     // To the left
268     if (theColumn > 0) {
269         if (isThereAMine(theRow, theColumn - 1)) { ret++; }
270     }
271     return ret;
272 }
273
274 private class Slot {
275
276     // Visited this turn?
277     private boolean visitedThisTurn = false;
278
279     // Does it have a mine under it?
280     private boolean hasMine = false;
281
282     // Still hidden?
283     private boolean hidden = true;
284
285     // Has the player made it suspicious?
286     private boolean suspicious = false;
287
288     // Marked with a question mark?
289     private boolean questionMark = false;
290
291     // Constructor

```

```

292     public Slot() {
293     }
294
295     // Getters
296     public boolean hasBeenVisitedThisTurn() {
297         return visitedThisTurn;
298     }
299     public boolean hasMine() {
300         return hasMine;
301     }
302     public boolean isHidden() {
303         return hidden;
304     }
305     public boolean isSuspicious() {
306         return suspicious;
307     }
308     public boolean hasQuestionMark() {
309         return questionMark;
310     }
311
312     // Setters
313     public void setHasBeenVisitedThisTurn(boolean hasBeenVisitedThisTurn) {
314         visitedThisTurn = hasBeenVisitedThisTurn;
315     }
316     public void setHasMine(boolean hasMine) {
317         this.hasMine = hasMine;
318     }
319     public void setIsHidden(boolean isHidden) {
320         hidden = isHidden;
321     }
322     public void setIsSuspicious(boolean isSuspicious) {
323         suspicious = isSuspicious;
324     }
325     public void setHasQuestionMark(boolean hasQuestionMark) {
326         questionMark = hasQuestionMark;
327     }
328 }
329 }
330
331 }

```

```

1 package minesweeper.logic;
2
3 public class BombException extends Exception {
4
5     private int row;
6     private int column;
7
8     /**
9      * Constructor
10     *
11     * @param theRow
12     *         Row of the bomb location
13     * @param theColumn
14     *         Column of the bomb location
15     */
16     public BombException(int theRow, int theColumn) {
17         row = theRow;
18         column = theColumn;
19     }
20
21     // Getters
22     public int getRow() {
23         return row;
24     }
25     public int getColumn() {
26         return column;
27     }
28 }

```

```

1 package minesweeper.gui;
2
3 import javax.swing.JFrame;
4 import javax.swing.BorderFactory;
5 import javax.swing.UIManager;

```



```

6
7 import java.awt.GridBagLayout;
8 import java.awt.GridBagConstraints;
9 import java.awt.Insets;
10 import java.awt.Color;
11
12 public class Main {
13
14     public Main() {
15     }
16
17     public static void main(String[] args) {
18         try {
19             UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
20         } catch (Exception e) {
21             e.printStackTrace();
22         }
23         // Main window, with a grid bag layout
24         GridBagLayout gridBagLayout1 = new GridBagLayout();
25         JFrame mainWindow = new JFrame();
26         mainWindow.getContentPane().setLayout(gridBagLayout1);
27
28         // Aspect and behavior of the main window
29         mainWindow.setLocation(200, 200);
30         mainWindow.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
31         mainWindow.setSize(570, 500);
32         mainWindow.setTitle("Minesweeper");
33
34         // Create a game panel
35         GamePanel pj = new GamePanel();
36         pj.setBorder(BorderFactory.createLineBorder(Color.WHITE));
37         mainWindow.getContentPane().add(pj,
38             new GridBagConstraints(0, 0, 1, 1, 1.0, 1.0,
39             GridBagConstraints.CENTER, GridBagConstraints.BOTH, new Insets(10, 10, 10, 10), 0, 0))
40         ↪ ;
41
42         // Show the panel (in a centered position)
43         mainWindow.setLocationRelativeTo(null);
44         mainWindow.setVisible(true);
45     }
46 }

```

```

1 package minesweeper.gui;
2
3 import java.awt.Color;
4 import java.awt.Component;
5 import java.awt.Font;
6 import java.awt.GridBagConstraints;
7 import java.awt.GridBagLayout;
8 import java.awt.HeadlessException;
9 import java.awt.Image;
10 import java.awt.Insets;
11 import java.awt.event.MouseEvent;
12 import java.awt.image.BufferedImage;
13 import java.awt.image.ColorModel;
14 import java.text.MessageFormat;
15 import java.util.List;
16
17 import javax.swing.BorderFactory;
18 import javax.swing.ImageIcon;
19 import javax.swing.JButton;
20 import javax.swing.JLabel;
21 import javax.swing.JOptionPane;
22 import javax.swing.JPanel;
23 import javax.swing.JTextField;
24 import javax.swing.border.Border;
25
26 import minesweeper.Constants;
27 import minesweeper.logic.BombException;
28 import minesweeper.logic.Map;
29
30 import org.apache.sanselan.common.ByteSources.ByteSourceInputStream;
31 import org.apache.sanselan.formats.gif.GifImageParser;
32
33 import com.jhlabs.image.MapColorsFilter;
34

```

```

35 public class GamePanel extends JPanel {
36
37     // Borders definition
38     private static final Border BEVEL_BORDER = BorderFactory.createRaisedBevelBorder();
39     private static final Border GRAY_BORDER = BorderFactory.createLineBorder(Color.LIGHT_GRAY);
40
41     // Mine colors
42     private static final String[] COLORS = { "blue", "green", "red", "pink", "#000099", "#996600",
43         ↪ " ", "#FF6600", "#99CC33" };
44
45     // HTML template for each mine
46     private static final String MINE_TEMPLATE = "<html><div style=\"font-size:12px;color:{0}\">"
47         ↪ + "{1}</div></html>";
48
49     protected GridBagLayout gridBagLayout1 = new GridBagLayout();
50
51     // Slots matrix
52     protected GraphicSlot[][] matrix;
53
54     // Graphic elements
55     protected JTextField box;
56     protected JButton resetButton;
57
58     // Can we play?
59     private boolean play = true;
60
61     // Marks the start of the game
62     protected boolean firstMove = true;
63
64     // Timestamp of the start of the game
65     protected long timestampGameStart = 0;
66
67     // Instances of the gif images
68     protected ImageIcon flag;
69     protected ImageIcon mine;
70     protected ImageIcon redMine;
71     protected ImageIcon questionMark;
72     protected ImageIcon smiley;
73     protected ImageIcon wrongFlag;
74
75     // Coordinates of the last mine found
76     private int rowLastMine = -1;
77     private int columnLastMine = -1;
78
79     // Any mine step on?
80     protected boolean mineStepOn = false;
81
82     // Map object instance
83     protected static Map map = null;
84
85     /**
86      * The Map is accesed through this method
87      *
88      * @return The <code>Map</code> of the application
89      */
90     private static Map theMap() {
91         if (map == null) {
92             map = new Map();
93         }
94         return map;
95     }
96
97     // Getters
98     public boolean canWePlay() {
99         return play;
100     }
101     public boolean isFirstMove() {
102         return firstMove;
103     }
104
105     // Timestamp
106     public long getGameStart() {
107         return timestampGameStart;
108     }
109
110     public boolean mineStepOn() {

```

```

109     return mineStepOn;
110 }
111
112 // Setters
113 public void setPlay(boolean areWePlaying) {
114     play = areWePlaying;
115 }
116 public void setFirstMove(boolean theFirstMove) {
117     firstMove = theFirstMove;
118 }
119 public void setGameStart(long millis) {
120     timestampGameStart = millis;
121 }
122 public void setStepOnMine(boolean isMineStepOn) {
123     mineStepOn = isMineStepOn;
124 }
125
126 // Gets an image off a file
127 private BufferedImage getImage(String path) {
128     BufferedImage ret = null;
129     try {
130         List images = new GifImageParser().getAllBufferedImages(new ByteSourceInputStream(
131             ↪ GamePanel
132             .class.getClassLoader().getResourceAsStream(path), path));
133
134         if (images != null && images.size() > 0) {
135             ret = (BufferedImage) images.get(0);
136         }
137     } catch (Exception e) {
138         e.printStackTrace();
139     }
140     return ret;
141 }
142
143 // Use color filter in order to get transparency
144 private Image filterImage(String path, Color filter) {
145     BufferedImage ret = null;
146
147     try {
148         BufferedImage tmp = getImage(path);
149
150         // This particular bit mask has transparent behavior
151         ↪ if (tmp != null) { MapColorsFilter f = new MapColorsFilter(filter.getRGB(), 0x00FFFFFF &
152             ↪ filter.getRGB());
153
154         // Make sure we obtain a standard RGB image
155         ret = f.createCompatibleDestImage(tmp, ColorModel.getRGBdefault());
156         f.filter(tmp, ret);
157
158         System.out.println(path);
159         System.out.println(tmp);
160         System.out.println(ret);
161     }
162     } catch (Exception e) {
163         e.printStackTrace();
164     }
165     return ret;
166 }
167
168 // Panel constructor
169 public GamePanel() {
170     try {
171         // Load the gif images
172         mine = new ImageIcon(filterImage(Constants.MINE_PATH, Color.WHITE));
173         flag = new ImageIcon(filterImage(Constants.FLAG_PATH, Color.WHITE));
174         redMine = new ImageIcon(getImage(Constants.RED_MINE_PATH));
175
176         questionMark = new ImageIcon(filterImage(Constants.QUESTION_MARK_PATH, Color.WHITE));
177         wrongFlag = new ImageIcon(filterImage(Constants.WRONG_FLAG_PATH, Color.WHITE));
178         smiley = new ImageIcon(filterImage(Constants.SMILEY_PATH, Color.WHITE));
179
180         // Build the dialog
181         jbInit();
182     }

```

```

183     catch (Exception e) {
184         e.printStackTrace();
185     }
186 }
187
188 protected void jbInit() throws Exception {
189     this.setLayout(gridBagLayout1);
190
191     // Matrix of JLabels
192     matrix = new GraphicSlot[Constants.ROWS][Constants.COLUMNS];
193     JTextField clock = new JTextField(Constants.ZERO);
194
195     clock.setEnabled(false);
196     clock.setHorizontalAlignment(JTextField.RIGHT);
197
198     Observer observer = new Observer(this, clock);
199     placeButtonsPanel(observer, clock);
200
201     // Place the JLabels
202     for (int row = 0; row < Constants.ROWS; row++) {
203         for (int column = 0; column < Constants.COLUMNS; column++) {
204
205             // Create the JLabel and get it into the panel
206             matrix[row][column] = new GraphicSlot(row, column);
207             matrix[row][column].addMouseListener(observer);
208             matrix[row][column].setMaximumSize(new java.awt.Dimension(Constants.SLOT_WIDTH,
↪ Constants.SLOT_HEIGHT));
209             matrix[row][column].setPreferredSize(new java.awt.Dimension(Constants.SLOT_WIDTH,
↪ Constants.SLOT_HEIGHT));
210
211             // The constraints are ok
212             this.add(matrix[row][column], new GridBagConstraints(column,
213                 row + 1, 1, 1, 1.0, 1.0, GridBagConstraints.CENTER,
214                 GridBagConstraints.BOTH, new Insets(0, 0, 0, 0), 0, 0));
215         }
216     }
217 }
218
219 protected void placeButtonsPanel(Observer observer, JTextField clock) {
220
221     // Reset button and counters
222     resetButton = new JButton(smiley);
223     resetButton.setName(Constants.RESET);
224     resetButton.addMouseListener(observer);
225
226     // Another panel to get a place in the grid for it
227     JPanel panelButtonCounter = new JPanel();
228     GridBagLayout layoutPanel = new GridBagLayout();
229     panelButtonCounter.setLayout(layoutPanel);
230
231     // Contains a button and a non editable box with the number of mines remaining
232     box = new JTextField();
233
234     // Add the lower panel
235     this.add(panelButtonCounter, new GridBagConstraints(0, 0,
236         Constants.COLUMNS, 1, 1.0, 2.0, GridBagConstraints.CENTER,
237         GridBagConstraints.BOTH, new Insets(10, 10, 10, 10), 0, 0));
238
239     // Add clock, buttons and mine counter to panel
240     panelButtonCounter.add(box, new GridBagConstraints(0, 0, 1, 1, 1.0,
241         1.0, GridBagConstraints.CENTER, GridBagConstraints.HORIZONTAL,
242         new Insets(5, 5, 5, 5), 0, 0));
243
244     panelButtonCounter.add(resetButton, new GridBagConstraints(1, 0, 1, 1,
245         8.0, 1.0, GridBagConstraints.CENTER, GridBagConstraints.NONE,
246         new Insets(5, 5, 5, 5), 0, 0));
247
248     panelButtonCounter.add(clock, new GridBagConstraints(2, 0, 1, 1, 1.0,
249         1.0, GridBagConstraints.CENTER, GridBagConstraints.HORIZONTAL,
250         new Insets(5, 5, 5, 5), 0, 0));
251
252     // The clock size
253     clock.setMaximumSize(new java.awt.Dimension(Constants.CLOCK_WIDTH, Constants.CLOCK_HEIGHT)
↪ );
254     clock.setPreferredSize(new java.awt.Dimension(Constants.CLOCK_WIDTH, Constants.
↪ CLOCK_HEIGHT));

```

```

255 // Boxes: format and color
256 box.setHorizontalAlignment(JTextField.RIGHT);
257 box.setText(Integer.toString(Constants.NUMBER_OF_MINES));
258 box.setSize(Constants.BOX_WIDTH, Constants.BOX_HEIGHT);
259 box.setEditable(false);
260 box.setPreferredSize(new java.awt.Dimension(Constants.BOX_WIDTH, Constants.BOX_HEIGHT));
261
262 // Colors
263 box.setBackground(Color.BLUE);
264 box.setForeground(Color.GRAY);
265 box.setBorder(BorderFactory.createLineBorder(Color.BLACK));
266
267 clock.setBackground(Color.BLUE);
268 clock.setForeground(Color.GRAY);
269 clock.setBorder(BorderFactory.createLineBorder(Color.BLACK));
270
271 }
272
273 // Coordinates of the last mine found
274 public void setLastMineCoordinates(int row, int column) {
275     rowLastMine = row;
276     columnLastMine = column;
277 }
278
279 /**
280  * Redraws the game map. Run through the map querying every position. If it
281  * is hidden, the color must be black. If it is revealed, the color is gray.
282  * If it is revealed, it must also write the number of mines around the slot.
283  *
284  * @param isAMine
285  *         Tells if we are painting the last turn, this meaning, if the
286  *         player has just stepped on a mine and we must reveal the game
287  *         panel situation
288  */
289 public void redrawGamePanel(boolean isAMine, int theRow, int theColumn) {
290     Map map = theMap();
291
292     // How many mines left?
293     box.setText(Integer.toString(map.remainingMines()));
294
295     // Run through the map
296     for (int row = 0; row < Constants.ROWS; row++) {
297         for (int column = 0; column < Constants.COLUMNS; column++) {
298
299             // If it is not hidden
300             GraphicSlot slot = matrix[row][column];
301             if (!map.isHidden(row, column)) {
302                 map.setHasBeenVisitedThisTurn(row, column, false);
303
304                 // Set background to gray
305                 slot.setBackground(Constants.VISIBLE_BACKGROUND);
306                 slot.setBorder(GRAY_BORDER);
307
308                 slot.setIcon(null);
309
310                 int numMines = map.getMinesAround(row, column);
311
312                 if (numMines > 0) {
313                     slot.setForeground(Color.BLUE);
314                 }
315                 else {
316                     slot.setForeground(Constants.VISIBLE_BACKGROUND);
317                 }
318                 slot.writeMinesNumber(numMines);
319             }
320
321             else if (isAMine) {
322                 // In this case, we paint a hidden slot in the last turn,
323                 // when the player just stepped on a mine
324                 paintHiddenSlot(slot, map.isSuspicious(row, column), true,
325                     false);
326             }
327
328             else if (map.isSuspicious(row, column)) {
329                 // Hidden and suspicious
330                 paintHiddenSlot(slot, true, false, false);
331             }
332         }
333     }

```

```

331     }
332
333     else if (map.hasQuestionMark(row, column)) {
334         // With a question mark
335         paintHiddenSlot(slot, false, false, true);
336     }
337
338     else {
339         // In this case, the slot returns to clean state
340         if (isFirstMove() || (slot.getRow() == theRow && slot.getColumn() == theColumn)) {
341
342             // Hidden and free of suspicion
343             paintHiddenSlot(slot, false, false, false);
344         }
345     }
346 }
347 }
348 }
349
350 // Paints a hidden slot in the panel
351 private void paintHiddenSlot(GraphicSlot slot, boolean isSuspicious,
352     boolean isMineStepOn, boolean hasQuestionMark) {
353
354     slot.setText(null);
355     slot.setBorder(BEVEL_BORDER);
356     slot.setBackground(Constants.HIDDEN_BACKGROUND);
357
358     // Was there a mine?
359     boolean isThereAMine = theMap().isThereAMine(slot.getRow(), slot.getColumn());
360
361     if (isMineStepOn && !isSuspicious) {
362         if (isThereAMine) {
363             if (slot.getRow() == rowLastMine && slot.getColumn() == columnLastMine) {
364                 slot.setIcon(redMine);
365             }
366
367             else {
368                 slot.setIcon(mine);
369             }
370         }
371
372         else {
373             slot.setForeground(Constants.HIDDEN_BACKGROUND);
374             slot.setText(Constants.EMPTY_SLOT_TEXT);
375             slot.setIcon(null);
376         }
377     }
378
379     else if (isMineStepOn && isSuspicious && !isThereAMine) {
380         slot.setIcon(wrongFlag);
381     }
382
383     else if (isSuspicious) {
384         slot.setIcon(flag);
385     }
386
387     else if (hasQuestionMark) {
388         slot.setIcon(questionMark);
389     }
390
391     else {
392         slot.setForeground(Constants.HIDDEN_BACKGROUND);
393         slot.setText(Constants.EMPTY_SLOT_TEXT);
394         slot.setIcon(null);
395     }
396 }
397 }
398
399 // Define JLabel with coordinates
400 protected class GraphicSlot extends JLabel {
401
402     // Properties
403     private int row;
404     private int column;
405
406     // New slot with black and white border

```

```

407 public GraphicSlot(int theRow, int theColumn) {
408     super();
409     row = theRow;
410     column = theColumn;
411     this.setOpaque(true);
412
413     // Every slot has a white border when just created
414     setBorder(BEVEL_BORDER);
415     setBackground(Constants.HIDDEN_BACKGROUND);
416
417     // Text font and alignment
418     setFont(new Font(Constants.FONT, Font.BOLD, Constants.FONT_SIZE));
419     this.setHorizontalAlignment(JLabel.CENTER);
420     setForeground(Constants.HIDDEN_BACKGROUND);
421     setText(Constants.EMPTY_SLOT_TEXT);
422
423     // Text: orange by default
424     setPreferredSize(new java.awt.Dimension(Constants.SLOT_WIDTH, Constants.SLOT_HEIGHT));
425 }
426
427 // Getters
428 public int getRow() {
429     return row;
430 }
431 public int getColumn() {
432     return column;
433 }
434
435 // Setters
436 public void setColor(Color color) {
437     setBackground(color);
438 }
439
440 // Nothing around, then nothing written
441 public void writeMinesNumber(int minesNumber) {
442     if (minesNumber != 0) {
443         setText(formatMinesNumber(minesNumber));
444         setFont(new Font(Constants.FONT, Font.BOLD, Constants.FONT_SIZE));
445     }
446     else {
447         setText(Constants.EMPTY_SLOT_TEXT);
448         setFont(new Font(Constants.FONT, Font.BOLD, Constants.FONT_SIZE));
449     }
450 }
451
452 /**
453  * @param minesNumber
454  *         Number of mines to paint in the slot
455  * @return A proper HTML string to render the number of mines around the slot
456  */
457 private String formatMinesNumber(int minesNumber) {
458     return MessageFormat.format(MINE_TEMPLATE, COLORS[minesNumber - 1], minesNumber);
459 }
460 }
461
462 // Events observer
463 // Every slot is subscribed to it
464 private class Observer extends java.awt.event.MouseAdapter {
465
466     // Panel
467     private GamePanel gamePanel = null;
468
469     // Field
470     private JTextField clock = null;
471
472     public Observer(GamePanel panel, JTextField look) {
473         super();
474         gamePanel = panel;
475         clock = look;
476
477         // The observer sets the clock to zero
478         ClockThread.reset(clock);
479     }
480
481     // Capture a click
482     public void mousePressed(MouseEvent e) {

```

```

483
484 // Mine step on?
485 boolean mine = gamePanel.mineStepOn();
486 Component c = e.getComponent();
487
488 try {
489     Map map = theMap();
490     if (c == null) {
491         return;
492     }
493
494     // If the event comes from a slot
495     if (c instanceof GraphicSlot && gamePanel.canWePlay()) {
496         manageSlotEvent(e, map, c);
497     }
498
499     // If it comes from a button
500     else {
501         String name = c.getName();
502         if (name != null && name.equals(Constants.RESET)) {
503
504             // Resetting the game
505             map.populateMap();
506             gamePanel.setPlay(true);
507             gamePanel.setStepOnMine(false);
508             gamePanel.setFirstMove(true);
509             mine = false;
510             clock.setText(Constants.ZERO);
511         }
512     }
513
514 } catch (BombException eb) {
515     gamePanel.setPlay(false);
516     gamePanel.setStepOnMine(true);
517     gamePanel.setLastMineCoordinates(eb.getRow(), eb.getColumn());
518
519     mine = true;
520     ClockThread.reset(clock);
521
522 } finally {
523     gamePanel.redrawGamePanel(mine, c instanceof GraphicSlot ? ((GraphicSlot) c).getRow()
524 ↪ : -1,
525                               c instanceof GraphicSlot ? ((GraphicSlot) c).getColumn() : -1);
526 }
527
528 // This method manages the click over the slots: is it a mine, is it suspicious...?
529 private void manageSlotEvent(MouseEvent e, Map map, Component c)
530     throws HeadlessException, BombException {
531
532     GraphicSlot slot = (GraphicSlot) c;
533     int row = slot.getRow();
534     int column = slot.getColumn();
535
536     if (e.getButton() == MouseEvent.BUTTON1) {
537
538         // Where did the event come from?
539         if (!map.isSuspicious(row, column)) {
540
541             map.click(row, slot.getColumn());
542         }
543
544         // Has the player won?
545         if (map.hasWon()) {
546             win();
547         }
548     }
549
550     else if (e.getButton() == MouseEvent.BUTTON3) {
551         // Marks the mines with a capital 'X'
552         if (map.isHidden(row, column)) {
553
554             if (map.isSuspicious(row, column)) {
555                 // Free of suspicion
556                 map.setFreeOfSuspicion(row, column);
557

```



```

558         // Mark with a question mark
559         map.addQuestionMark(row, column);
560     }
561
562     else if (map.hasQuestionMark(row, column)) {
563         // Clear it
564         map.clearQuestionMark(row, column);
565     }
566
567     else {
568         // Suspicious
569         map.setSuspicious(row, column);
570     }
571
572     // Has the player won?
573     if (map.hasWon()) {
574         win();
575     }
576 }
577
578 // Counter
579 if (gamePanel.isFirstMove()) {
580     gamePanel.setGameStart(System.currentTimeMillis());
581     gamePanel.setFirstMove(false);
582     ClockThread.startGame();
583 }
584 }
585 }
586
587 // Calculates the game time
588 private void win() throws HeadlessException {
589
590     // Stop the clock; then paint the message
591     ClockThread.reset(clock);
592     gamePanel.setPlay(false);
593
594     long now = System.currentTimeMillis();
595     long time = now - gamePanel.getGameStart();
596
597     StringBuffer message = new StringBuffer(Constants.YOU_HAVE_FINISHED_IN);
598
599     message.append(Math.round(time / 1000.0));
600     message.append(Constants.SECONDS);
601
602     JOptionPane.showMessageDialog(gamePanel, message.toString(), Constants.CONGRATULATIONS,
603     ↪ JOptionPane.PLAIN_MESSAGE);
604 }
605 }

```

```

1 package minesweeper.gui;
2
3 import javax.swing.JTextField;
4 import javax.swing.SwingUtilities;
5
6 public class ClockThread extends Thread {
7
8     // A second expressed in milliseconds
9     private static final long SECOND_TIME_MILLIS = 1000;
10
11     // Single instance of the thread
12     private static ClockThread instance;
13
14     // Seconds counter
15     private long seconds = 0;
16
17     // Reference to the graphic element with the seconds that have passed since the start of the
18     ↪ game
19     private JTextField clock = null;
20
21     private ClockThread(JTextField look) {
22         clock = look;
23         seconds = 0;
24     }
25
26     // Resets the counter

```

```

26 public static void reset(JTextField look) {
27     if (instance != null) {
28         // Tries to interrupt the current thread. If any problem raises,
29         // we get an InterruptedException; if not, we just change our state
30         instance.interrupt();
31     }
32
33     // Leave the former instance for the garbage collector
34     instance = null;
35
36     // Create a new instance
37     instance = new ClockThread(look);
38 }
39
40 // This one will make the thread start the counter
41 public static void startGame() {
42     instance.start();
43 }
44
45 // Will just update the clock every XXX milliseconds
46 public void run() {
47     try {
48
49         // The thread will continue, unless interrupted from the outside
50         while (!interrupted()) {
51
52             // Paint the time in the GUI (using invokeLater)
53             SwingUtilities.invokeLater(new Runnable() {
54
55                 @Override
56                 public void run() {
57                     clock.setText(Long.toString(seconds++));
58                 }
59             });
60
61             // Wait for a second
62             sleep(SECOND_TIME_MILLIS);
63         }
64     } catch (Exception e) {
65         e.printStackTrace();
66     }
67 }
68
69 }

```

```

1 package minesweeper;
2
3 import java.awt.Color;
4
5 public interface Constants {
6
7     // Paths for the images (gif)
8     String QUESTION_MARK_PATH = "questionMark.gif";
9     String SMILEY_PATH = "smiley.gif";
10    String WRONG_FLAG_PATH = "wrongFlag.gif";
11    String FLAG_PATH = "flag.gif";
12    String MINE_PATH = "mine.gif";
13    String RED_MINE_PATH = "redMine.gif";
14
15    // Font for the numbers shown
16    String FONT = "Dialog";
17
18    // Start again
19    String RESET = "Start_again";
20
21    // Slot marked as suspicious by the user
22    char X = 'x';
23
24    // Font size
25    int FONT_SIZE = 7;
26
27    // Width and height for the box with the remaining mines information
28    int BOX_WIDTH = 55;
29    int BOX_HEIGHT = 20;
30
31    // Reset the clock

```

```

32 String ZERO = "0";
33
34 // Slots: width and height
35 int SLOT_WIDTH = 15;
36 int SLOT_HEIGHT = 15;
37
38 // Empty slot content
39 String EMPTY_SLOT_TEXT = "V";
40
41 // Messages
42 String CONGRATULATIONS = "Congratulations!";
43 String YOU_HAVE_FINISHED_IN = "You_have_finished_it_in_";
44 String SECONDS = "_seconds!";
45
46 // Background colors
47 Color HIDDEN_BACKGROUND = Color.LIGHT_GRAY;
48 Color VISIBLE_BACKGROUND = Color.WHITE;
49
50 // Clock width and height
51 int CLOCK_WIDTH = 40;
52 int CLOCK_HEIGHT = 20;
53
54 // Map rows and columns
55 int ROWS = 16;
56 int COLUMNS = 30;
57
58 int NUMBER_OF_MINES = 20;
59 }

```